上海理工大学光电信息与计算机工程学院

**《操作系统》作业报告**

****

**专　　业 智能科学与技术**

**学生姓名　　高浩琦**

**学　 号 2035060413**

**年　　级 2020**

**指导教师 魏赟**

**成 绩：**

**教师签字：**

目录

[实验一 进程调度 3](#_Toc467495852)

[实验二 存储管理调度算法 4](#_Toc467495853)

[实验三 资源分配算法 5](#_Toc467495854)

# 作业一 进程调度

**一、作业目的**

深入了解进程调度中，时间片轮转以及抢占优先级两组算法的运行情况

以及提高自身编程能力

**二、作业软硬件要求**

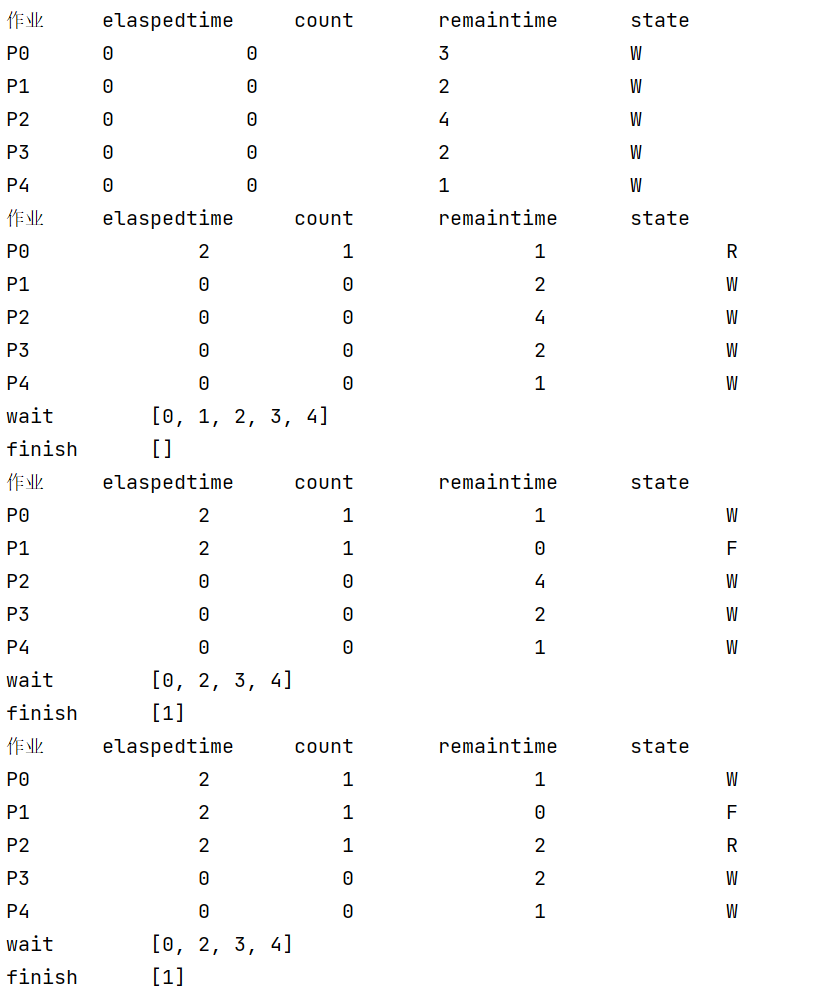
Java文档，以及idea编辑坏境

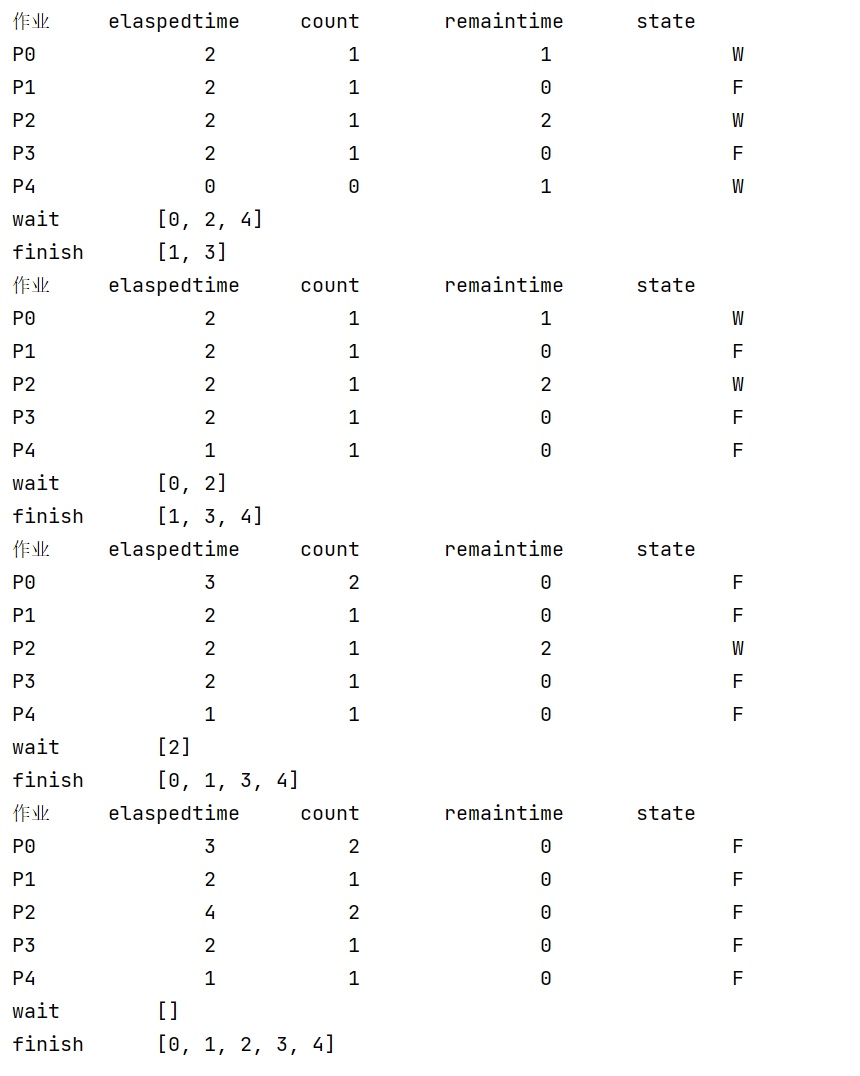
**三、作业内容及结果（按要求，若需编程的需写出源程序和运行程序后的结果截图）**

**1，时间片轮转**

import java.util.\*;  
class RR{  
 int remaintime,elaspedtime=0,count=0,servetime;  
 String state="W";  
 public RR(int servetime){  
 this.remaintime=servetime;  
 this.servetime=servetime;  
 }  
}  
public class schedule extends NullPointerException {  
 public static void main(String[] args) {  
 int timeslice=2;  
   
 *//设置各个进程的运行时间* RR[]p=new RR[5];  
 p[0]=new RR(3);  
 p[1]=new RR(2);  
 p[2]=new RR(4);  
 p[3]=new RR(2);  
 p[4]=new RR(1);  
   
 *//定义完成队列finishquery；等待队列waitquery；定义job队列* HashSet<Integer> finishedquery=new HashSet<>();  
 HashSet<Integer> waitline=new HashSet<>();  
 LinkedList <RR> job= new LinkedList<>();  
 job.add(p[0]);  
 job.add(p[1]) ;  
 job.add(p[2]);  
 job.add(p[3]);  
 job.add(p[4]);  
   
 *//首次输出未运行的进程队列* System.*out*.println("作业\t\telaspedtime\t\tcount\t\tremaintime\t\tstate");  
 for (int i=0;i<job.size();i++){  
 System.*out*.println("P"+i+"\t\t"+p[i].elaspedtime+"\t\t\t"+p[i].count+"\t\t\t\t"+p[i].remaintime+"\t\t\t\t"+p[i].state);  
 }  
   
 *//程序运行部分* for (int m=0;m< job.size();m++){  
 for (int i=0;i< job.size();i++){  
 if (p[i].state.equals("F"))  
 {continue;}  
 p[i].remaintime-=timeslice;  
 p[i].elaspedtime+=timeslice;  
 p[i].state="R";  
 p[i].count++;  
 if (p[i].remaintime<=0)  
 {p[i].state="F";p[i].elaspedtime=p[i].servetime;p[i].remaintime=0;  
 finishedquery.add(i);}  
 System.*out*.println("作业\t\telaspedtime\t\tcount\t\tremaintime\t\tstate");  
 for (int j=0;j< job.size();j++)  
 { System.*out*.println("P"+j+"\t\t\t\t"+p[j].elaspedtime+"\t\t\t"+p[j].count+"\t\t\t\t"+p[j].remaintime+"\t\t\t\t"+p[j].state);}  
 if(p[i].remaintime!=0)  
 {p[i].state="W";}  
 for (int j=0;j< job.size();j++){  
 if (p[j].remaintime!=0)  
 waitline.add(j);  
 }  
 System.*out*.println("wait\t\t"+waitline);  
 System.*out*.println("finish\t\t"+finishedquery);  
 for (int j=0;j< job.size();j++){  
 waitline.remove(j);  
 }  
 }  
 }  
 }  
 }

运行截图

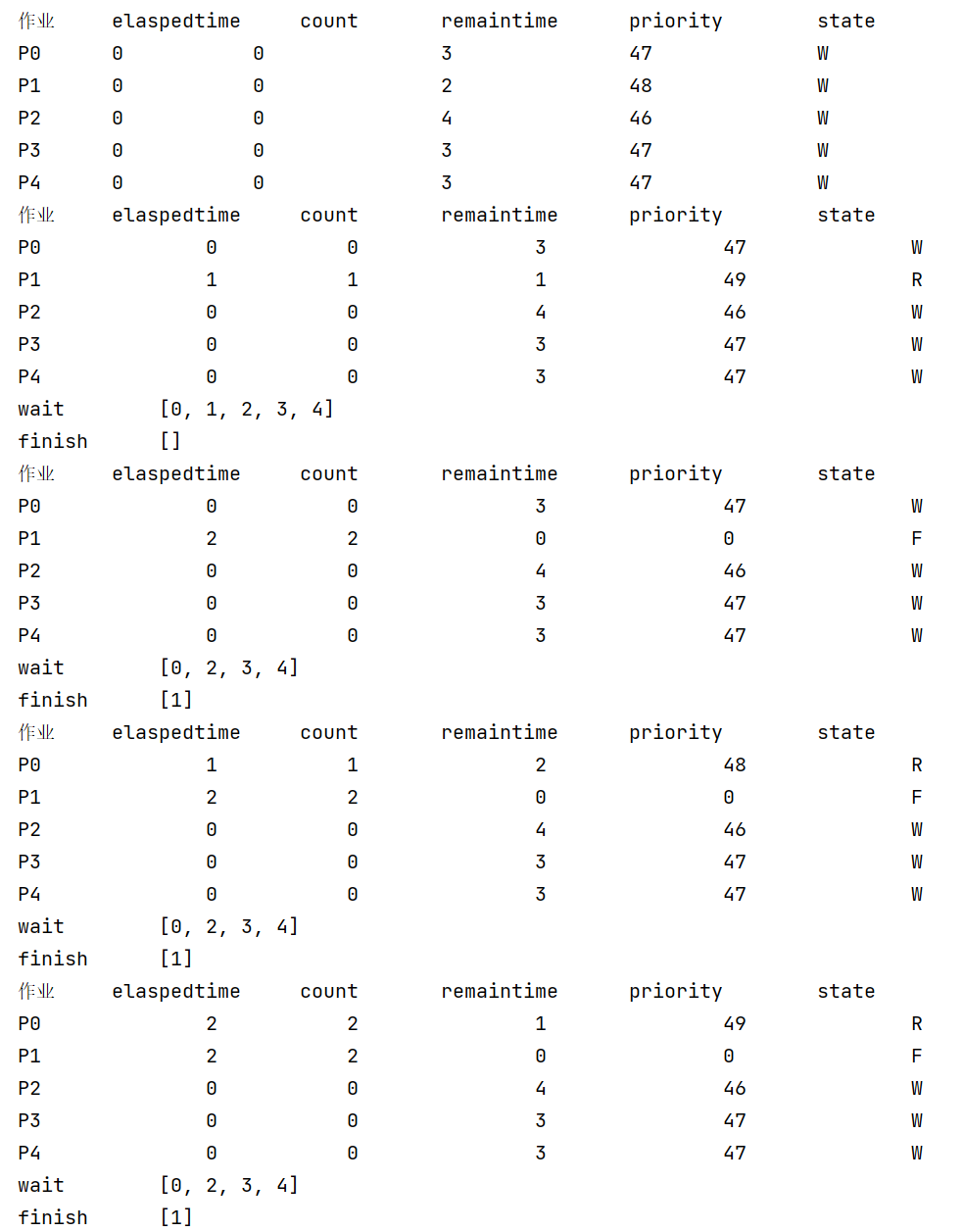


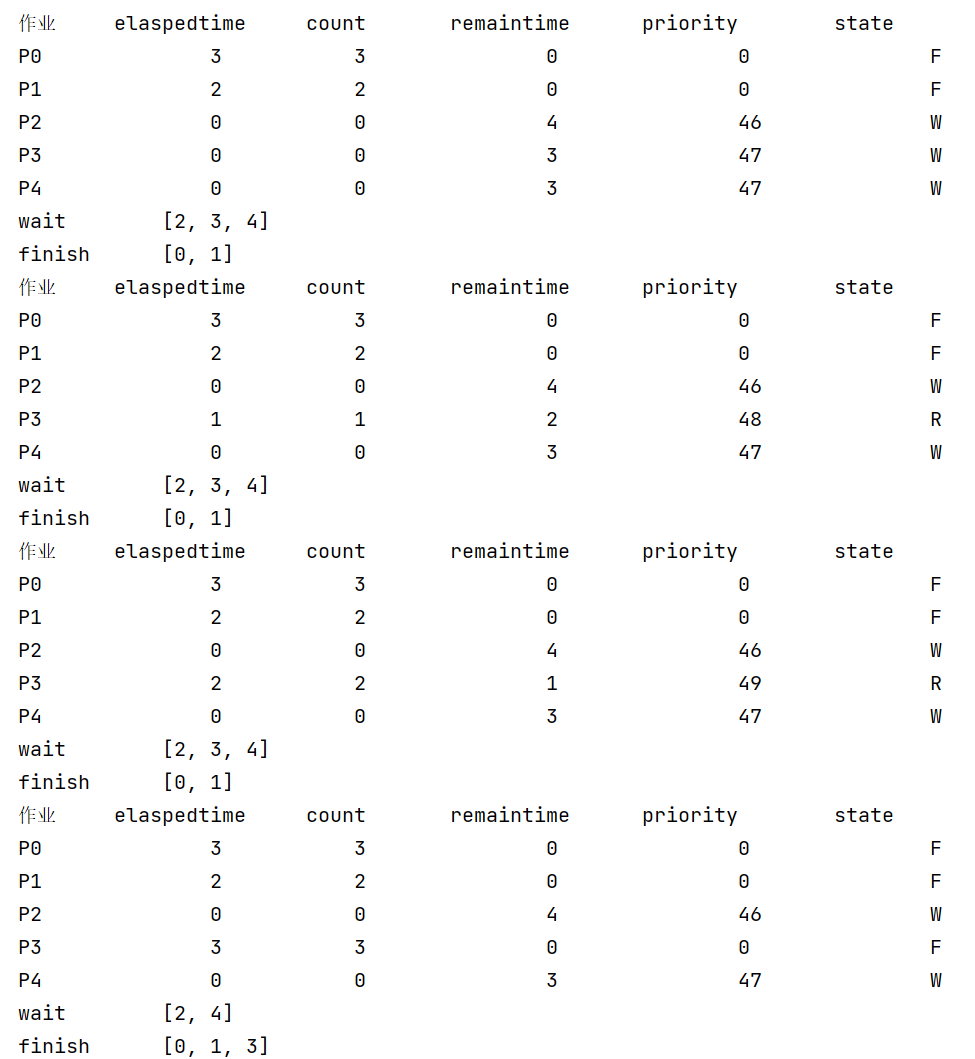


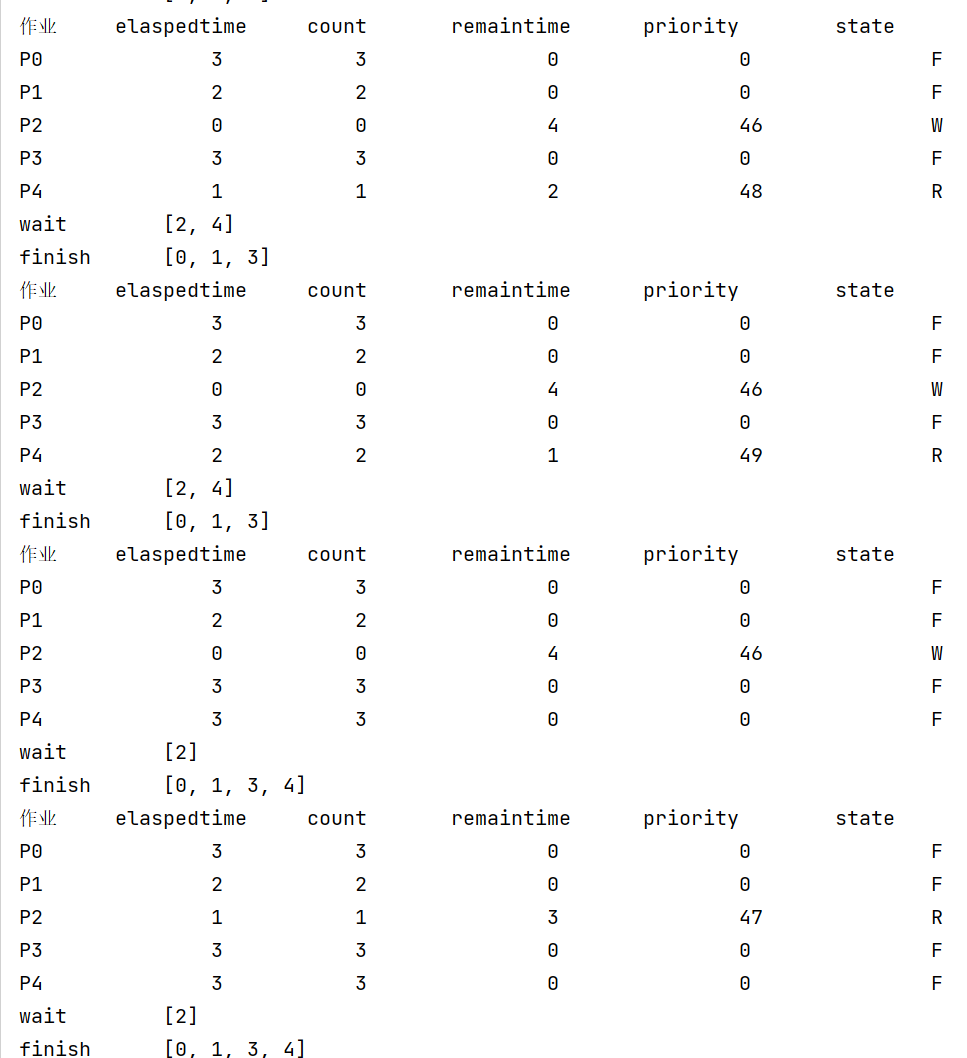
**2.抢占优先级**

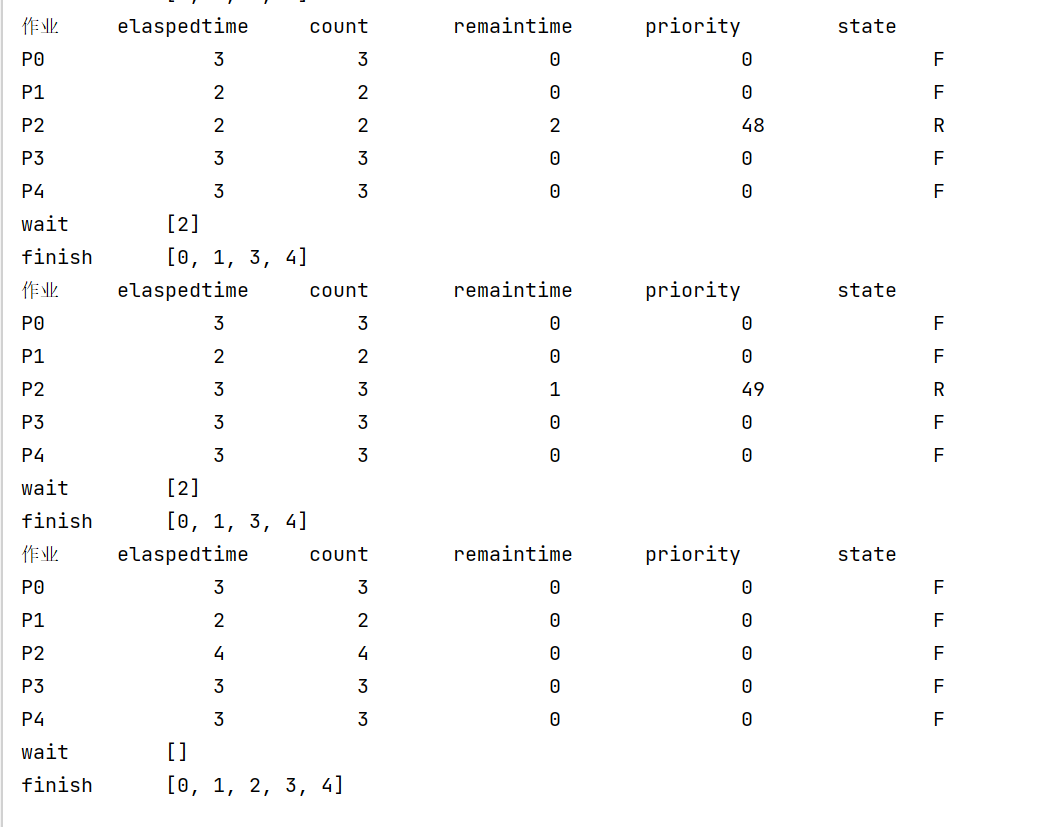
import java.util.HashSet;  
import java.util.LinkedList;  
  
class PRR{  
 int remaintime,elaspedtime=0,count=0,servetime,priority;  
 String state="W";  
 public PRR(int servetime){  
 this.remaintime=servetime;  
 this.servetime=servetime;  
 this.priority=50-this.remaintime;  
 }  
 public boolean input(PRR[] a){  
 for (int i=0;i<5;i++){  
 if (a[i].remaintime!=0){  
 return true;  
 }  
 }  
 return false;  
 }  
}  
public class pschedule {  
 public static void main(String[] args) {  
 int timeslice=1;  
  
 *//设置各个进程的运行时间* PRR[] P=new PRR[5];  
 P[0]=new PRR(3);  
 P[1]=new PRR(2);  
 P[2]=new PRR(4);  
 P[3]=new PRR(3);  
 P[4]=new PRR(3);  
  
 *//定义完成队列finishquery；等待队列waitquery；定义job队列* HashSet<Integer> pfinishedquery=new HashSet<>();  
 HashSet<Integer> pwaitline=new HashSet<>();  
 LinkedList<PRR> job=new LinkedList<>();  
 job.add(P[0]);  
 job.add(P[1]);  
 job.add(P[2]);  
 job.add(P[3]);  
 job.add(P[4]);  
  
 *//首次输出未运行的进程队列* System.*out*.println("作业\t\telaspedtime\t\tcount\t\tremaintime\t\tpriority\t\tstate");  
 for (int i=0;i<job.size();i++){  
 System.*out*.println("P"+i+"\t\t"+P[i].elaspedtime+"\t\t\t"+P[i].count+"\t\t\t\t"+P[i].remaintime+"\t\t\t\t"+P[i].priority+"\t\t\t\t"+P[i].state);  
 }  
 int tem=P[0].priority;  
  
 *//程序运行部分* while (P[0].input(P)) {  
 for (int i = 0; i < job.size(); i++) {  
 if (P[i].priority > tem)  
 tem = P[i].priority;  
 }  
 for (int j = 0; j < job.size(); j++) {  
 if (tem == P[j].priority) {  
 if (P[j].state.equals("F")) {  
 continue;  
 }  
 P[j].remaintime -= timeslice;  
 P[j].elaspedtime += timeslice;  
 P[j].state = "R";  
 P[j].count++;  
 P[j].priority = 50 - P[j].remaintime;  
 if (P[j].remaintime <= 0) {  
 P[j].state = "F";  
 P[j].elaspedtime = P[j].servetime;  
 P[j].remaintime = 0;  
 P[j].priority = 0;  
 pfinishedquery.add(j);  
 }  
 System.*out*.println("作业\t\telaspedtime\t\tcount\t\tremaintime\t\tpriority\t\tstate");  
 for (int n = 0; n < job.size(); n++) {  
 System.*out*.println("P" + n + "\t\t\t\t" + P[n].elaspedtime + "\t\t\t" + P[n].count + "\t\t\t\t" + P[n].remaintime + "\t\t\t\t" + P[n].priority + "\t\t\t\t" + P[n].state);  
 }  
 if (P[j].remaintime != 0) {  
 P[j].state = "W";  
 }  
 for (int m = 0; m < job.size(); m++) {  
 if (P[m].remaintime != 0)  
 pwaitline.add(m);  
 }  
 System.*out*.println("wait\t\t" + pwaitline);  
 System.*out*.println("finish\t\t" + pfinishedquery);  
 for (int m = 0; m < job.size(); m++) {  
 pwaitline.remove(j);  
 }  
 break;  
 }  
 }  
 tem=0;  
 }  
 }  
}

运行截图









**四、自评（评价是否完成实验要求及自己的收获）**

基本圆满完成进程调度模块的上机实验后,自身对进程调度算法的更加深入了解，也了解了一些计算机软件的运行情况，并对自己编程的能力有了一定的提高

# 作业二 存储管理调度算法

**一、作业目的**

深入了解操作系统中存储管理分区分配算法，清晰如何建立分区描述区，如何针对不同的放置策略建立相应队列结构，并编写分区分配算法和分区回收算法

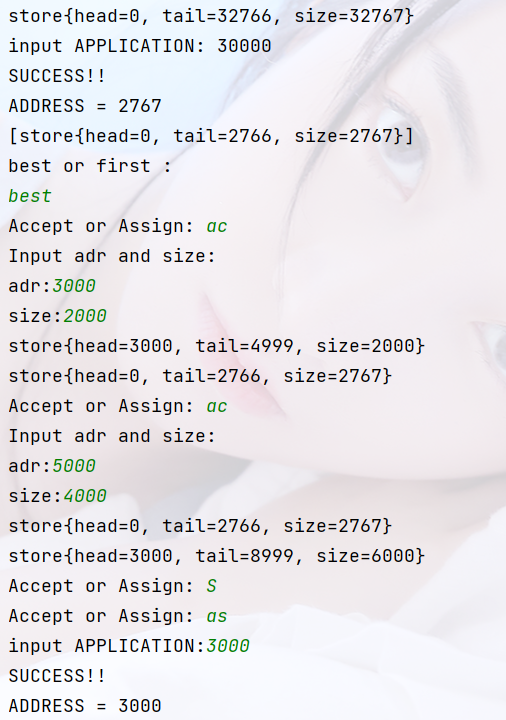
**二、作业软硬件要求**

….

**三、作业内容及结果（按要求，若需编程的需写出源程序和运行程序后的结果截图）**

package com.company;  
  
  
public class store{  
  
 private int head;  
 private int tail;  
 private int size;  
  
 public store(int head, int tail, int size) {  
 this.head = head;  
 this.tail = tail;  
 this.size = size;  
 }  
  
  
 public store(int head, int size) {  
 this.head = head;  
 this.tail = head + size - 1 ;  
 this.size = size;  
 }  
  
  
 @Override  
 public String toString() {  
 return "store{" +  
 "head=" + head +  
 ", tail=" + tail +  
 ", size=" + size +  
 '}';  
 }  
  
 public int getHead() {  
 return head;  
 }  
  
 public void setHead(int head) {  
 this.head = head;  
 }  
  
 public int getTail() {  
 return tail;  
 }  
  
 public void setTail(int tail) {  
 this.tail = tail;  
 }  
  
 public int getSize() {  
 return size;  
 }  
  
 public void setSize(int size) {  
 this.size = size;  
 }  
}

package com.company;  
  
import com.sun.source.tree.WhileLoopTree;  
  
import java.util.\*;  
import java.util.ArrayList;  
import java.util.Collections;  
import java.util.Comparator;  
import java.util.List;  
  
public class best {  
  
 public static void main(String[] args) {  
 *// write your code here* store bigstore = new store(0,32766,32767);  
  
 int putadd =0;  
*// System.out.println();* System.*out*.println(bigstore.toString());  
  
  
 System.*out*.println("input APPLICATION: 30000");  
 if (bigstore.getSize() > 30000){  
  
 System.*out*.println("SUCCESS!!");  
  
 putadd = bigstore.getSize()-30000;  
 System.*out*.println("ADDRESS = " + putadd);  
 }else {  
 System.*out*.println("FALSE!! TOO LARGE!!");  
 }  
  
 Scanner sc = new Scanner(System.*in*);  
  
  
*// bigstore.getTail()* List <store> stores = new ArrayList<>();  
  
*// List <store> pri\_stores = new ArrayList<>();* stores.add(new store(0,2766,2767));  
  
 System.*out*.println(stores.toString());  
  
 System.*out*.println("best or first :");  
  
 String way = sc.next();  
  
 boolean inway = way.equals("best");  
  
 while (true){  
  
 System.*out*.print("Accept or Assign: ");  
  
 String choice = sc.next();  
  
 if (choice.equals("ac")){  
  
 System.*out*.println("Input adr and size:");  
 System.*out*.print("adr:");  
 int adr = sc.nextInt();  
 System.*out*.print("size:");  
 int size = sc.nextInt() ;  
  
 int nil = adr + size - 1;  
 boolean loop = true;  
  
 for (store store : stores) {  
  
 if ((store.getHead() >= nil && store.getHead() <= adr)  
 ||(store.getTail() <= nil && store.getTail() >= adr)  
 ||(store.getHead() < adr && store.getTail() > nil)){  
  
 loop = false;  
 System.*out*.println("cross the border!");  
 break;  
 }  
 if (store.getTail()+1 == adr){  
 store.setSize(store.getSize()+size);  
 store.setTail(store.getTail()+size);  
 loop = false;  
  
 }  
 if (store.getHead() -1 == nil){  
 store.setHead(adr);  
*// store.setTail();* store.setSize(store.getSize() + size);  
 loop = false;  
 }  
  
  
*// if(store.g)* }  
  
 if (loop){  
  
 stores.add(new store(adr,size));  
  
 }  
  
 if (inway) {  
 stores.sort(new Comparator<store>() {  
 @Override  
 public int compare(store o1, store o2) {  
  
 Integer a1 = o1.getSize();  
 Integer a2 = o2.getSize();  
 return a1.compareTo(a2);  
  
 }  
  
 });  
 }  
  
  
*// for (store store : pri\_stores) {  
// System.out.println(store.toString());  
// }  
//* for (store store : stores) {  
 System.*out*.println(store.toString());  
 }  
  
 }  
  
  
 if (choice.equals("as")){  
 System.*out*.print("input APPLICATION:");  
  
 int application = sc.nextInt();  
 boolean loop = true;  
  
 for (store store : stores) {  
  
 if (store.getSize() >= application){  
  
 store.setSize(store.getSize() - application);  
 store.setTail(store.getTail() - application);  
 System.*out*.println("SUCCESS!!");  
 System.*out*.println("ADDRESS = " + store.getSize());  
 loop = false;  
 break;  
*// store.setTail(store.getTail() - application);* }  
 }  
  
 if (loop){  
  
 System.*out*.println("TOO large application");  
 }  
  
  
 if (inway) {  
 stores.sort(new Comparator<store>() {  
 @Override  
 public int compare(store o1, store o2) {  
  
 Integer a1 = o1.getSize();  
 Integer a2 = o2.getSize();  
 return a1.compareTo(a2);  
 }  
 });  
 }  
  
  
 for (store store : stores) {  
 if (store.getSize() > 0) {  
 System.*out*.println(store.toString());  
 }  
 }  
 }  
 if (choice.equals("EOF")){  
 break;  
 }  
 }  
 }  
}



**四、自评（评价是否完成实验要求及自己的收获）**

# 作业三 资源分配算法

**一、作业目的**

实现银行家算法

实现安全性算法

**二、作业软硬件要求**

Java文档，以及idea编辑坏境

**三、作业内容及结果（按要求，若需编程的需写出源程序和运行程序后的结果截图）**

package com.company.Bankers;  
  
public class banker {  
 public static final int *resoureNum* = 5;  
 public static final int *n* = 5;  
 public static final int *processNum* = 3;  
 public static final int *m* = 3;  
  
 static int[] *available* = { 3, 3, 2 }; *// 可用资源* static int[][] *max* = { { 7, 5, 3 },  
 { 3, 2, 2 },  
 { 9, 0, 2 },  
 { 2, 2, 2 },  
 { 4, 3, 3 } }; *// 最大需求量* static int[][] *allocation* = { { 0, 1, 0 },  
 { 2, 0, 0 },  
 { 3, 0, 2 },  
 { 2, 1, 1 },  
 { 0, 0, 2 } }; *// 已分配的资源* static int[][] *need* = { { 7, 4, 3 },  
 { 1, 2, 2 },  
 { 6, 0, 0 },  
 { 0, 1, 1 },  
 { 4, 3, 1 } };; *// 需求矩阵* static int[] *request* = new int[*processNum*];  
  
  
  
*// public Queue list;* public static void main(String[] args) {  
  
  
 System.*out*.println("judge the system security");  
 if (*securityAlgorithm*()) {  
*// System.out.println("t0时刻是安全的");* System.*out*.println("SYSTEM SECURITY!!!");  
 } else {  
 System.*out*.println("t0时刻是不安全的");  
 }  
  
 *// p1请求资源* System.*out*.println("============");  
 System.*out*.println(" p1 = 1 0 2 judge the request security");  
 *request*[0] = 1;  
 *request*[1] = 0;  
 *request*[2] = 2;  
*// request= [1,0,2;];  
 bankerAlgorithm*(*request*, 1);  
  
 *// p4请求资源* System.*out*.println("============");  
 System.*out*.println(" p4 = 3 3 0 judge the request security");  
 *request*[0] = 3;  
 *request*[1] = 3;  
 *request*[2] = 0;  
 *bankerAlgorithm*(*request*, 4);  
  
 *// p0请求资源* System.*out*.println("============");  
 System.*out*.println(" p1 = 0 2 0 judge the request security");  
 *request*[0] = 0;  
 *request*[1] = 2;  
 *request*[2] = 0;  
 *bankerAlgorithm*(*request*, 0);  
  
 *// p0请求资源  
//  
// System.out.println("============");  
// System.out.println("如果银行家算法中吧p0发出的向量请求改为request(0,1,0)");  
// request[0] = 0;  
// request[1] = 1;  
// request[2] = 0;  
// bankerAlgorithm(request, 0);* }  
  
  
  
  
 *//request 1,0,2* public static void bankerAlgorithm(int request[], int i) {  
  
  
 *// 步骤1  
 // 打印当前即将比较的request，need，的一些信息  
  
// System.out.println("进程" + i + "执行请求：开始运行");* System.*out*.print("request ");  
 *printOneMa*(request);  
 System.*out*.println();  
 System.*out*.print("need" + i + " ");  
 *printOneMa*(*need*[i] );  
 System.*out*.println();  
  
  
 if (*compare*(request, *need*[i])) {  
 System.*out*.println("request<=need");  
 *// 步骤2  
 // 打印当前即将比较的request，available，的一些信息* System.*out*.print("request" + " ");  
 *printOneMa*(request);  
 System.*out*.println();  
 System.*out*.print("available" + " ");  
 *printOneMa*(*available*);  
 System.*out*.println();  
  
 if (*compare*(request, *available*)) {  
 System.*out*.println("request<=available");  
 *// 步骤3  
 available* = *oneMatrixSub*(*available*, request);  
 *allocation*[i] = *oneMatrixAdd*(*allocation*[i], request);  
 *need*[i] = *oneMatrixSub*(*need*[i], request);  
  
 *// 步骤4  
 // 调用安全性算法* if (*securityAlgorithm*()) {  
*// System.out.println("此次资源分配后，系统处于安全状态，允许请求，将资源分配给" + i + "进程");* System.*out*.println("SYSTEM SECURITY!!!");  
  
 System.*out*.print("available = ");  
 *printOneMa*(*available*);  
 System.*out*.println();  
 } else {  
 System.*out*.println("RESOURCE INSUFFICIENT!!! p" + i + " CAN NOT OBTAIN RESOURCES IMMEDIATELY.");  
  
 *available* = *oneMatrixAdd*(*available*, request);  
 *allocation*[i] = *oneMatrixSub*(*allocation*[i], request);  
 *need*[i] = *oneMatrixAdd*(*need*[i], request);  
  
 System.*out*.println(i + "进程处于等待状态！");  
 System.*out*.print("available = ");  
 *printOneMa*(*available*);  
 System.*out*.println();  
 }  
 } else {  
 System.*out*.println("request>available");  
 System.*out*.println("RESOURCE INSUFFICIENT!!! p" + i + " CAN NOT OBTAIN RESOURCES IMMEDIATELY.");  
 }  
 } else {  
 System.*out*.println("request>need");  
 System.*out*.println("所需要资源数，已经超出了进程" + i + "宣布的最大值（request>need）");  
 }  
 }  
  
  
 public static boolean securityAlgorithm() {  
 *// 步骤1  
 // 初始化一些变量  
 // int[] work = available; 错误的赋值方法，这样的话，下面程序对work进行赋值操作，改变了availabe,  
 // 重新定义一个一维数组，将available的值赋给这个以为数组work* System.*out*.println("-----安全性算法执行：-----");  
 System.*out*.println(" " + "Work " + "Need " + "Allocation " + " Work + Allocation" );  
  
  
 int[] work = new int[*m*];  
  
 for (int i = 0; i < work.length; i++) {  
  
 work[i] = *available*[i];  
  
 }  
  
 boolean[] finish = new boolean[*n*];  
  
 for (int i = 0; i < finish.length; i++) {  
 finish[i] = false;  
 }  
 int count = 0; *// 计数器，用于判断是否finish[n]里所有Boolean都是true  
  
 // 步骤2* for (int i = 0; i < *n*; i++) {  
 if (*compare*(*need*[i], work) && finish[i] == false) *// 找到满足这个条件的进程i* {  
 *// 步骤3* System.*out*.print("p" + i + " ");  
 *printOneMa*(work);  
 *printOneMa*(*need*[i]);  
 *printOneMa*(*allocation*[i]);  
  
 work = *oneMatrixAdd*(work, *allocation*[i]);  
  
*// System.out.print(" " + "Work+Allocation:");  
  
 printOneMa*(work);*// 打印work+allocation* System.*out*.println();  
 finish[i] = true;  
 *// go to step2  
 // 因为for循环内的部分执行后，会有i++操作，所以为了保证i从0开始，设置i=-1,i++ = 0;* if (i == *n* - 1) {  
 i = -1;  
 }  
 }  
 }  
 *// 步骤4* for (int j = 0; j < finish.length; j++) {  
 if (finish[j] == true) {  
 count++;  
 }  
 }  
  
  
 return count == *n*;  
 }  
  
  
  
  
*// public static void main(String[] args) {  
//  
//// resoureNum  
//  
//  
//  
//  
// }* public static int[] oneMatrixSub(int[] first, int[] second) {  
 for (int i = 0; i < first.length; i++) {  
 first[i] = first[i] - second[i];  
 }  
 return first;  
 }  
  
 */\*\*  
 \* 写一个一维数组矩阵加法函数；  
 \*  
 \* @param first  
 \* 一维数组  
 \* @param second  
 \* 一维数组  
 \* @return 一维数组  
 \*/* public static int[] oneMatrixAdd(int[] first, int[] second) {  
 for (int i = 0; i < first.length; i++) {  
 first[i] = first[i] + second[i];  
 }  
 return first;  
 }  
  
 */\*\*  
 \* 写一个二维数组矩阵加法函数；  
 \*  
 \* @param a  
 \* 二维数组  
 \* @param b  
 \* 二维数组  
 \* @return 二维数组  
 \*/* public int[][] twoMatrixAdd(int[][] a, int[][] b) {  
 for (int i = 0; i < b.length; i++) {  
 for (int j = 0; j < b[i].length; j++) {  
 a[i][j] = a[i][j] + b[i][j];  
 }  
 }  
 return a;  
 }  
  
 */\*\*  
 \* 写一个二维数组矩阵减法函数；  
 \*  
 \* @param a  
 \* 二维数组  
 \* @param b  
 \* 二维数组  
 \* @return 二维数组  
 \*/* public static int[][] twoMatrixSub(int[][] a, int[][] b) {  
 for (int i = 0; i < b.length; i++) {  
 for (int j = 0; j < b[i].length; j++) {  
 a[i][j] = a[i][j] - b[i][j];  
 }  
 }  
 return a;  
 }  
  
*// /\*\*  
// \* 矩阵比较函数，判断两个矩阵每个相应位置上的数是否都大于另一个矩阵,如果第一个小于等于第二个返回true,否则为false  
// \*  
// \* @param a  
// \* 一维数组  
// \* @param b  
// \* 一维数组  
// \* @return boolean  
// \*/* public static boolean compare(int[] first, int[] second) {  
 int count = 0;  
 for (int i = 0; i < first.length; i++) {  
 if (first[i] <= second[i]) {  
 count++;  
 }  
 }  
  
 return count == first.length;  
 }  
  
 */\*\*  
 \* 打印一维数组内容  
 \*  
 \* @param temp  
 \* 一维数组  
 \*/* public static void printOneMa(int[] temp) {  
 for (int i = 0; i < temp.length; i++) {  
 System.*out*.print(temp[i]);  
 System.*out*.print(" ");  
 }  
 System.*out*.print(" ");  
 }  
  
}





**四、自评（评价是否完成实验要求及自己的收获）**